Refatoração da Extensão NetLogo de Aprendizagem por Reforço para Integração com a Biblioteca BURLAP

Eloísa Bazzanella[1], **Matheus M. Barros**[1], **Fernando Santos**[1]

[1]Departamento de Engenharia de Software
Universidade do Estado de Santa Catarina (UDESC)
Ibirama – SC – Brasil

{elobazzanella,matheusmbarros01}@gmail.com, fernando.santos@udesc.br

Abstract. *Agent-based modeling and simulation is a simulation paradigm that allows focusing on individuals, their interactions, and the resulting complex behavior. Agent-based simulations are typically developed in simulation environments that provide agent-related features. One such environment is NetLogo, for which there is an extension that enables the use of reinforcement learning by agents, specifically the Q-Learning algorithm. This paper describes the refactoring of this extension to integrate with BURLAP, a mature reinforcement learning Java library. Evaluations have shown that the refactoring has maintained the consistency and functionality of the NetLogo extension. With the refactoring, the NetLogo extension now enables the use of reinforcement learning algorithms SARSA(λ) and Actor-Critic, in addition to Q-Learning.*

Resumo. *A modelagem e simulação baseada em agentes é um paradigma de simulação que permite focar nos indivíduos, em suas interações e no comportamento complexo resultante. Simulações baseadas em agentes são geralmente desenvolvidas em ambientes de simulação que fornecem recursos relacionados a agentes. Um desses ambientes é o NetLogo, que possui uma extensão para que os agentes possam utilizar aprendizagem por reforço, especificamente o algoritmo Q-Learning. Este artigo descreve a refatoração dessa extensão para integração com a BURLAP, uma biblioteca Java consolidada de aprendizagem por reforço. Avaliações evidenciaram que a refatoração manteve a extensão NetLogo consistente e funcional. A partir da refatoração, a extensão NetLogo passa a permitir o uso dos algoritmos de aprendizagem por reforço SARSA(λ) e Actor-Critic, além do Q-Learning.*

1. Introdução

As simulações baseadas em agentes (SBAs) são uma ferramenta poderosa para modelar e entender fenômenos complexos em diversas áreas do conhecimento. De acordo com [Epstein e Axtell 1996], essas simulações consistem na criação de modelos realistas e dinâmicos que capturam a interação e o comportamento dos agentes em um sistema. Portanto, as SBAs constituem uma ferramenta precisa para compor e testar ocorrências de diversos fenômenos e analisar suas implicações.

Agentes são sistemas inteligentes, capazes de perceber o ambiente, autônomos em sua tomada de decisão e aptos a interação com outros agentes. Para a tomada de decisão,

um agente pode se basear em deduções lógicas, assim como o raciocínio humano, ou por meio da dedução combinada a algum mecanismo [Wooldridge 2009]. Sendo assim, os agentes e suas interações são os principais alvos de estudo quando se fala em SBAs.

SBAs podem ser implementadas em linguagens de propósito geral, como Python. No entanto, já existem ambientes de simulação que fornecem recursos específicos e simplificam o desenvolvimento. Um destes é o NetLogo, um ambiente de programação projetado para modelagem de sistemas complexos. Segundo [Wilensky 1999], o NetLogo permite a criação de SBAs com facilidade e rapidez, bem como a análise e visualização dos resultados das simulações. Recentemente foi disponibilizada uma extensão (biblioteca) para o NetLogo que permite utilizar aprendizagem por reforço em SBAs [Kons 2019]. Uma limitação dessa extensão é que suporta apenas o algoritmo *Q-Learning*.

Este artigo descreve a refatoração da extensão NetLogo existente. O objetivo é integrar a extensão com a BURLAP [MacGlashan et al. 2018], uma biblioteca de aprendizagem por reforço Java consolidada, e disponibilizar outros algoritmos de aprendizagem por reforço. A partir da refatoração, a extensão agora permite utilizar, além do *Q-Learning*, também os algoritmos *SARSA(λ)* e *Actor-Critic*.

Uma avaliação foi realizada para evidenciar que a refatoração manteve a estabilidade da extensão NetLogo, sem introduzir inconsistências nos comandos existentes e nos resultados do algoritmo *Q-Learning* já disponível. Por fim, o *SARSA(λ)* e *Actor-Critic* foram utilizados em duas SBA para evidenciar o funcionamento da extensão refatorada.

2. Fundamentação Teórica

2.1. Simulações Baseadas em Agentes

Simulações baseadas em agentes (SBAs) têm sido utilizadas para simular sistemas complexos e dinâmicos, nos quais um conjunto de agentes autônomos interagem entre si e com o ambiente [Macal e North 2010]. As SBAs têm sido amplamente utilizado em diversas áreas para compreender e prever o comportamento de sistemas complexos, tais como em economia, ciências sociais, biologia, e engenharia [Bonabeau 2002]. SBAs permitem criar modelos mais realistas e precisos, uma vez que levam em consideração a heterogeneidade, a adaptação e a interação dos agentes com o ambiente [Macy e Willer 2002].

Para a criação de uma SBA é necessário definir as características dos agentes, como seu comportamento, objetivos e regras de interação. Uma vez definidos esses aspectos, é possível utilizar algoritmos de aprendizagem de máquina, para permitir que os agentes aprendam e se adaptem ao ambiente em que estão inseridos.

Além disso, as SBAs permitem a realização de experimentos virtuais que seriam inviáveis ou impraticáveis na vida real [Epstein e Axtell 1996]. Dessa forma, os modelos baseados em agentes podem ser utilizados para testar diferentes cenários, avaliar a eficácia de estratégias de tomada de decisão e propor soluções para problemas complexos.

2.2. Aprendizagem por Reforço

A aprendizagem por reforço (*reinforcement learning — RL*) é uma área de estudo da inteligência artificial que busca desenvolver algoritmos capazes de aprender a tomar decisões em situações complexas a partir do *feedback* de um ambiente externo [Sutton e Barto 2018]. A RL é uma abordagem indicada para situações em que

se deseja encontrar uma política de comportamento ótima. Uma política especifica como o agente deve agir em qualquer situação (estado) que possa se deparar. A política ótima, por sua vez, é a política que resulta no melhor desempenho do agente. A política ótima pode ser aprendida a partir das interações com o ambiente, mediante o conhecimento do estado do agente, das ações efetuadas e das mudanças nos estados [Sutton e Barto 2018].

Entre as características da RL, é possível mencionar a aprendizagem baseada na ação do agente no ambiente, que resulta num valor de recompensa, usado para tomar decisões posteriores. Outra característica importante é o *delayed reward*, que diz respeito à qualidade das ações, isto é, um valor de recompensa alto em determinado momento não significa necessariamente que a ação tomada é a recomendada, pois o intuito do agente é alcançar objetivos globais e não locais. Ademais, a RL também caracteriza-se por ser orientada à objetivo, ou seja, o agente não precisa conhecer detalhes da modelagem do ambiente; ele simplesmente age no ambiente tentando alcançar um objetivo [Sutton e Barto 2018]. Dentre os algoritmos de RL disponíveis estão o *Q-Learning*, o *SARSA(λ)*, e o *Actor-Critic*. Estes algoritmos buscam maximizar a recompensa esperada do agente no ambiente. Sobre estes algoritmos, a seguir serão apresentados aspectos relevantes ao presente trabalho. Recomenda-se que o leitor interessado em se aprofundar nos detalhes destes algoritmos consulte [Russel e Norvig 2004] e [Sutton e Barto 2018].

No *Q-Learning* [Watkins e Dayan 1992] e no *SARSA(λ)* [Rummery e Niranjan 1994], o agente aprende uma função de ação-valor. Tal função, denominada $Q(s, a)$, estima a qualidade de cada ação a em cada estado s do ambiente. Estes algoritmos se baseiam na ideia de que a melhor ação a ser tomada em um determinado estado é aquela que maximiza o valor de Q. Os valores de $Q(s, a)$ são determinados iterativamente, a medida que o agente atua no ambiente. A cada ação executada, o ambiente fornece ao agente um sinal de recompensa r. Desta forma, após experienciar o ambiente, tanto o *Q-Learning* quanto o *SARSA(λ)* estimam os valores de $Q(s, a)$ a partir das recompensas obtidas ao longo do tempo. A diferença entre o *Q-Learning* e o *SARSA(λ)* é que o último é um algoritmo *on-policy*, enquanto o primeiro é *off-policy*. Em um algoritmo *on-policy* o agente segue uma política de comportamento definida, e a aprendizagem atua para refinar essa política. Já em um algoritmo *off-policy* o agente não segue uma política de comportamento definida, sendo livre para experienciar ações arbitrariamente durante a aprendizagem [Sutton e Barto 2018].

Por fim, o *Actor-Critic* [Konda e Tsitsiklis 2000] combina as estratégias de funcionamento do *Q-Learning* e do *SARSA(λ)*, resultando em uma estratégia de RL mais eficiente. O *Actor-Critic* possui duas etapas. A etapa de avaliação de política diz respeito ao uso eficiente da experiência já aprendida pelo agente. Enquanto a etapa de melhoria da política serve para melhorar a política em todas as etapas até a convergência [Peters e Schaal 2008]. Em outras palavras, no *Actor-Critic* a etapa de avaliação pode ser interpretada como um elemento ator, que realiza a aproximação da função de utilidade dos estados (estratégia do *SARSA(λ)*). Já a etapa de melhoria pode ser interpretada como um elemento crítico, que realiza a aproximação da função de utilidade das ações (estratégia do *Q-Learning*) [Wang et al. 2007].

2.3. NetLogo e a Extensão Q-Learning

O NetLogo é um ambiente programável para modelagem e simulação de fenômenos naturais e sociais [Wilensky 1999]. Ele oferece uma linguagem de programação própria, com

comandos de alto nível para implementar funcionalidades de SBAs. Com isso, é possível dar instruções para centenas de agentes independentes que agem simultaneamente, com o intuito de explorar as conexões entre os comportamentos de nível micro dos indivíduos e os padrões de nível macro que resultam de suas interações [Sklar 2007]. Estatísticas indicam que em 2020, 33.6% das simulações disponíveis no repositório CoMSES eram feitas em NetLogo [CoMSES 2020].

A partir da versão 2.0.1, o NetLogo passou a fornecer uma API para a criação de extensões. Uma extensão é um módulo NetLogo que estende sua linguagem de programação para prover comandos e recursos adicionais. A API disponibiliza diversos elementos para o desenvolvedor utilizar durante a implementação, facilitando a integração com a linguagem NetLogo [Tisue e Wilensky 2004].

Recentemente, [Kons 2019] desenvolveu e disponibilizou uma extensão para uso do algoritmo *Q-Learning* no NetLogo. O objetivo desta extensão é auxiliar no processo de criação de agentes inteligentes, possibilitando que o desenvolvedor de simulações não precise implementar o *Q-Learning* manualmente. Para isso, a extensão disponibiliza novos comandos que, quando utilizados, definem os elementos requeridos pelo *Q-Learning*. A Figura 1 apresenta os comandos disponíveis na extensão, que podem ser classificados em comandos de *configuração* e de *execução* da aprendizagem.

```
; comandos de configuracao da aprendizagem
qlearningextension:state-def ; especificar os estados
qlearningextension:actions ; especificar as acoes
qlearningextension:reward ; especificar a recompensa
qlearningextension:learning-rate ; especificar a taxa de aprendizagem
qlearningextension:discount-factor ; especificar o fator de desconto
qlearningextension:action-selection ; especificar estrategia de selecao da acao

; comandos de execucao da aprendizagem
qlearningextension:learning ; executar acao e aprender
qlearningextension:act ; apenas executar uma acao
qlearningextension:learn ; apenas aprender a partir da acao executada e recompensa obtida
```

Figura 1. Comandos da Extensão NetLogo. Fonte: [Kons 2019]

Os comandos de *configuração* permitem que o desenvolvedor especifique o problema de aprendizagem, formado pelos estados, ações, recompensas, e também os parâmetros do algoritmo. Para usar o *Q-Learning*, basta ao desenvolvedor utilizar estes comandos no *setup* da simulação. Já os comandos de *execução* permitem ativar a aprendizagem na implementação do *step* da simulação, fazendo com que o agente execute o algoritmo *Q-Learning* considerando o estado, ação, recompensa, e os parâmetros do algoritmo. A extensão e a documentação de seus comandos estão disponíveis online.[1]

[Bazzanella e Santos 2021] avaliaram o benefício que a extensão *Q-Learning* fornece para o desenvolvimento de SBAs. Por meio de uma avaliação quantitativa, evidenciaram que o tamanho do código fonte das simulações é significativamente reduzido quando se utiliza a extensão para implementar o *Q-Learning*. Apesar dessa vantagem, atualmente a extensão se limita a disponibilizar apenas o algoritmo *Q-Learning* para RL.

[1] https://github.com/agentbasedsimulations/qlearning-netlogo-extension

2.4. Biblioteca BURLAP

A *Brown-UMBC Reinforcement Learning and Planning* (BURLAP) é uma biblioteca Java para desenvolvimento de agentes inteligentes que usam aprendizagem por reforço e planejamento [MacGlashan et al. 2018]. A BURLAP estabelece uma estrutura geral para que seja possível definir um domínio de problema, e fornece uma ampla variedade de algoritmos RL integrados, domínios pré-fabricados e ferramentas de visualização. Dentre os algoritmos disponibilizados estão o *Q-Learning*, o *SARSA(λ)*, e o *Actor-Critic*. Além disso, ela fornece uma interface de programação simples para desenvolver novos algoritmos de aprendizagem e planejamento. Com o uso da BURLAP, torna-se mais fácil implementar e testar algoritmos de RL em diversos tipos de problemas [MacGlashan et al. 2018].

A BURLAP tem sido amplamente utilizada em pesquisas acadêmicas em aprendizagem por reforço e planejamento, como em estudos sobre controle de robôs e jogos eletrônicos. Além disso, a biblioteca também tem sido usada para desenvolver aplicações práticas em áreas como sistemas autônomos e robótica. A BURLAP está disponível online[2]. Para utilizá-la, basta realizar as interfaces Java disponibilizadas para implementar o problema de aprendizagem em questão. Há interfaces para implementar os estados, as ações, o ambiente, e as transições que ocorrem no ambiente quando as ações são executadas. Uma vez instanciadas e integradas, pode-se aplicar os algoritmos disponibilizados pela BURLAP para realizar a aprendizagem do agente no problema em questão.

3. Refatoração da Extensão

O objetivo da refatoração da extensão NetLogo de aprendizagem por reforço é utilizar a estrutura e algoritmos de RL fornecidos pela biblioteca BURLAP. Esta seção descreve as alterações realizadas na extensão para utilizar a BURLAP e disponibilizar, além do *Q-Learning*, outros dois algoritmos de RL, o *SARSA(λ)* e o *Actor-Critic*.

Inicialmente, foi necessário realizar a adaptação da extensão proposta por [Kons 2019], que estava originalmente implementada em Scala, para a linguagem Java, visando a sua integração com a biblioteca BURLAP. Essa adaptação foi essencial devido à natureza da biblioteca BURLAP, escrita em Java, com o objetivo de assegurar a compatibilidade e viabilizar uma integração eficiente entre as duas partes.

Para incorporar a BURLAP na extensão, foi especificada uma arquitetura de integração entre a BURLAP e a simulação no NetLogo. Essa integração se utiliza dos recursos disponibilizados pela API do NetLogo[3] para acessar elementos da simulação. A partir dos comandos NetLogo disponibilizados pela extensão, pode-se instanciar os objetos da BURLAP necessários para implementar a aprendizagem nos agentes. Estes objetos, por estarem vinculados aos elementos da simulação, permitem que a BURLAP acesse as informações necessárias à aprendizagem, tais como estados, ações e recompensas.

A arquitetura projetada para fazer a integração entre a biblioteca BURLAP e a simulação NetLogo é apresentada na Figura 2. Os elementos em azul são disponibilizados pela BURLAP, no caso, as interfaces a serem implementadas. Já os elementos em vermelho são as classes disponibilizadas pela API do NetLogo. As classes da extensão, na cor branca, são responsáveis por criar comandos para serem utilizados no NetLogo,

[2] http://burlap.cs.brown.edu

[3] https://github.com/NetLogo/NetLogo/wiki/Extensions-API

Figura 2. Arquitetura da Extensão NetLogo refatorada com a BURLAP

afim de receber e armazenar as informações da simulação e implementar o problema de aprendizagem utilizando a BURLAP. A seguir são explicadas as classes desta arquitetura.

A `ExtensionCommands` é responsável por determinar quais comandos serão disponibilizados no NetLogo. Ela herda as funcionalidades da `DefaultClassManager`, exigência da API de integração para permitir ao ambiente NetLogo reconhecer os comandos disponibilizados e seus argumentos. Um exemplo de comando mostrado no diagrama da Figura 2 é a `LearningRateCommand`, que especifica a sintaxe e a execução do comando que define a taxa de aprendizagem. Todas as outras classes de comando seguem a mesma estrutura, realizando a interface `Command`. Se um comando retorna algum valor, é implementado como um `Reporter`.

A `Session` segue o padrão *singleton* para armazenar a instância da extensão e os agentes envolvidos, que são objetos do tipo `AgentLearning`. A `AgentLearning` é responsável por todas as informações necessárias para a aprendizagem, tais como fator de desconto, taxa de aprendizagem, função de recompensa e informações do objeto `Agent` fornecido pela API do NetLogo.

A `StateDefinition` define os estados, armazenando as informações que serão usadas na sua especificação. Já a `ActionSelection` seleciona a próxima ação, armazenando as informações sobre a estratégia de seleção de ação e seus parâmetros. Ambas são agregadas à `AgentLearning`. A `Reward` implementa a interface `RewardFunction` e é responsável por gerenciar as recompensas do agente de acordo com seu estado atual. A `IsEndEpisode` implementa a interface `TerminalFunction` fornecida pela BURLAP e determina se um estado é terminal.

A `AgentState` implementa a interface `MutableState` da BURLAP e converte as informações sobre os estados do objeto `StateDefinition` para uso na BURLAP. Já a `AgentStateModel` implementa a interface `FullStateModel` da BURLAP e faz a transição de estados após a execução da ação selecionada.

Por fim, as classes `QLearningAlgorithm`, `SarsaAlgorithm` e `ActorCriticAlgorithm` têm a responsabilidade de iniciar a criação da aprendizagem e executá-la usando os recursos fornecidos pela BURLAP. Cada uma delas implementa a interface `DomainGenerator` e tem uma única instância, com dois métodos principais: para configurar a aprendizagem; e para executar o processo de aprendizagem.

Tendo em vista que a extensão foi ampliada para incorporar outros algoritmos de RL além do *Q-Learning*, foi necessário renomeá-la, passando a se chamar `learningextension`. Além disso, para a incorporação desses novos algoritmos, foi necessário criar novos comandos NetLogo para serem executados no *setup* da simulação. A Figura 3 apresenta os novos comandos incorporados.

```
learningextension:define-algorithm ; "qlearning", "sarsa-lambda" or "actor-critic"
learningextension:lambda
learningextension:setup
```

Figura 3. Novos comandos disponibilizados na extensão.

O comando `learningextension:define-algorithm` permite especificar qual algoritmo será utilizado na aprendizagem (*Q-Learning*, *SARSA(λ)*, ou *Actor-Critic*).

O comando `learningextension:lambda` define a taxa de exigibilidade, parâmetro requerido pelos algoritmos *SARSA(λ)* e *Actor-Critic*. Por fim, o comando `learningextension:setup` deve ser utilizado para instanciar os objetos da BURLAP internamente na extensão e deixá-la pronta para executar o processo de aprendizagem. A documentação de todos os comandos da extensão refatorada está disponível online.[4].

O diagrama de atividades da Figura 4 descreve o fluxo de integração da extensão refatorada (raia central) com a simulação NetLogo e a biblioteca BURLAP. Essa integração ocorre através da arquitetura apresentada anteriormente na Figura 2. Conforme mencionado na seção 2.3, os comandos da extensão podem ser classificados em comandos de *configuração* e de *execução* da aprendizagem.

O ponto de início `A` representa a chamada de algum comando de *configuração* (por exemplo `learningextension:state-def`). A classe `StateDefinitionCommand` é responsável por validar a sintaxe e processar o comando utilizando-se da API do NetLogo. Nessa classe são definidas as variáveis que serão consideradas para especificar o estado do problema de aprendizagem. Depois de validada a sintaxe, é iniciado o processamento do comando. Neste processamento é instanciada a classe `StateDefinition` com as informações recebidas e este objeto é vinculado ao objeto `AgentLearning`, que também foi instanciado. A classe `AgentLearning` centraliza todas as informações relevantes para o aprendizado do agente. Todos os demais comandos de *configuração* seguem a mesma lógica de receber a informação por meio de uma classe que implementa a API do NetLogo e salva essas informações nos objetos da extensão refatorada.

Para finalizar a configuração da aprendizagem, o desenvolvedor utiliza o comando `learningextension:setup`, representado pelo ponto de início `B`. Ao ser executado, chama a classe `LearningCommand` que verifica a sintaxe e usa todas as informações que estão salvas no objeto `AgentLearning` para chamar a classe de *learning*. Essa classe varia conforme o algoritmo que foi selecionado, podendo ser o `QLearningAlgorithm`, `SarsaAlgorithm` ou `ActorCriticAlgorithm`. Nessa classe que é feita a inicialização do aprendizado na BURLAP, ou seja, a modelagem do problema.

Por fim, o ponto de início `C` representa a chamada do comando `learningextension:learning`, que é um comando de *execução* da aprendizagem. Este comando deve ser chamado dentro do procedimento que implementa o *step* da simulação. Ao utilizá-lo, é feita a chamada para a classe `LearningCommand`, que valida a sintaxe e reconhece o algoritmo que está sendo utilizado. Ao saber o algoritmo que está sendo utilizado, chama a classe de *learning* e executa os passos do aprendizado.

4. Avaliação da Extensão Refatorada

Com o intuito de avaliar o funcionamento da extensão, foram implementadas duas SBAs para verificar a aprendizagem de cada algoritmo. A hipótese é que, se o agente aprender uma política consistente com o que cada algoritmo se propõe, então a refatoração da extensão com integração do NetLogo e BURLAP está consistente e funcional.

A primeira SBA é o cenário *Cliff Walking* de [Sutton e Barto 2018], apresentado na Figura 5a. Neste cenário o ambiente é um mundo bidimensional dividido em células. O agente sempre ocupa uma única célula e pode se locomover por entre as células com

[4]https://github.com/agentbasedsimulations/qlearning-netlogo-extension/tree/burlap-migration

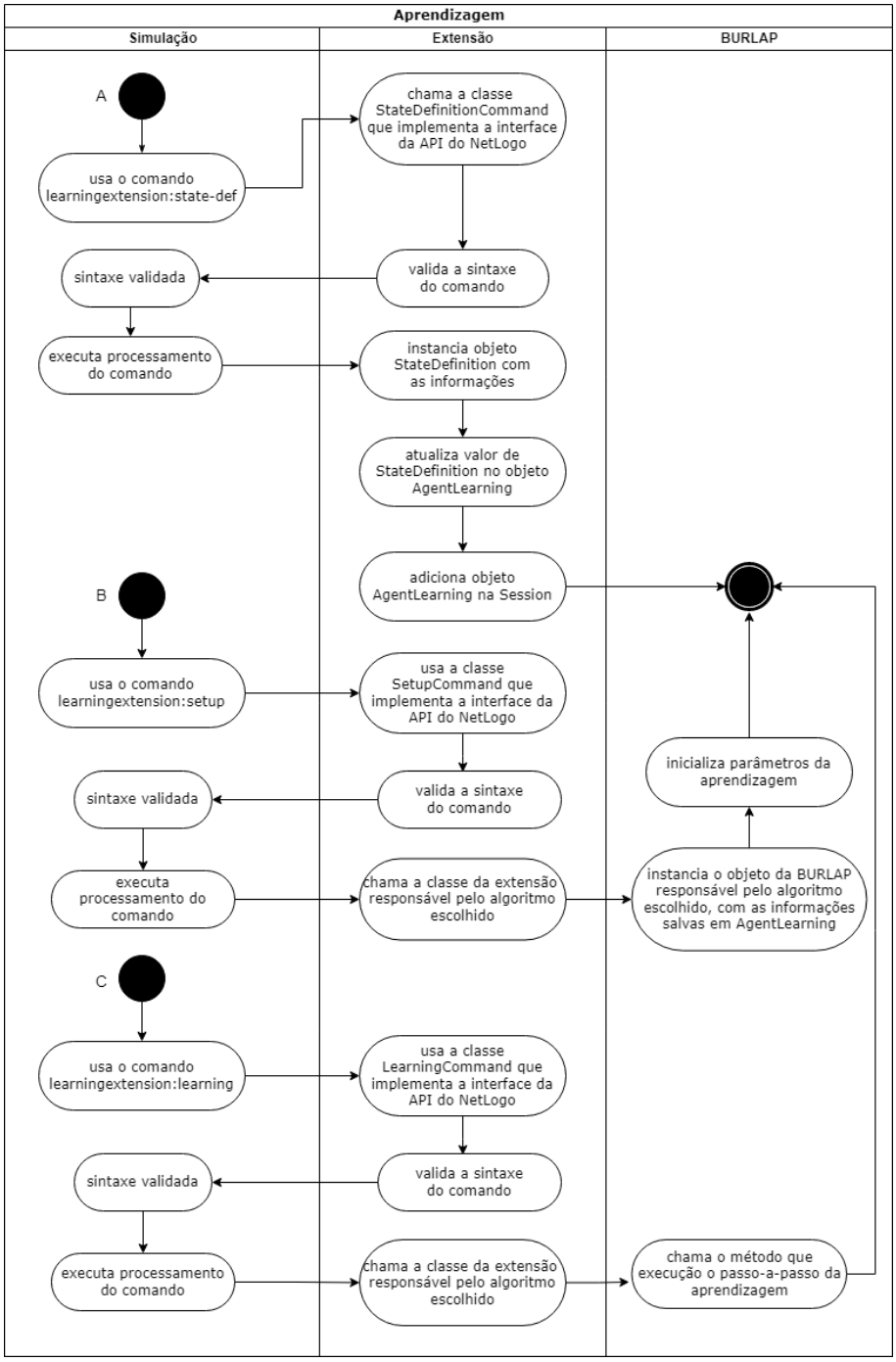
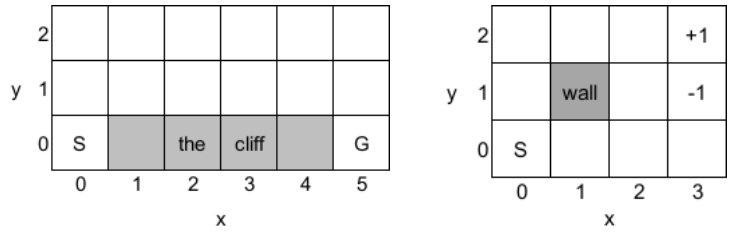


Figura 4. Diagrama de Integração NetLogo e BURLAP

as ações de ir para: cima, baixo, esquerda e direita. O estado do agente, portanto, é representado pela sua posição na grade de células. Neste cenário o agente deve aprender a se locomover, para sair do ponto S e chegar ao ponto G, traçando o menor percurso possível ou o mais seguro, sem passar pelo penhasco, representado pelas células em cinza. Quando o agente se desloca para qualquer célula que não seja o penhasco, recebe uma recompensa no valor -1. Se cair no penhasco recebe recompensa de -100.

A segunda SBA é o cenário *Mundo 4x3* de [Russel e Norvig 2004]. Neste cenário

o ambiente também é um mundo bidimensional dividido em células, apresentado na Figura 5b. As ações do agente, bem como a representação dos estados, são as mesmas do cenário *Cliff Walking*. Há dois estados terminais, sendo que um oferece recompensa +1 e outro fornece recompensa -1. Qualquer outra célula fornece ao agente uma recompensa no valor de -0.04 (exceto a célula inacessível ao agente indicada em cinza). O agente deve aprender a se locomover do ponto S até o estado terminal com máxima recompensa.



(a) Cliff Walking **(b)** Mundo 4x3

Figura 5. Simulações utilizadas para avaliação da extensão refatorada

Para avaliar a aprendizagem proporcionada pelos algoritmos *Q-Learning* e *SARSA(λ)*, foram coletados os valores estimados da função $Q(s, a)$ para cada par estado/ação (s, a) do problema. Já no caso do algoritmo *Actor-Critic*, foram coletados os valores de utilidade de cada estado s do problema. A Tabela 1 apresenta a média destes valores na SBA *Cliff Walking*, considerando 5 execuções por algoritmo. Os parâmetros adotados para cada algoritmo foram: *Q-Learning*: $\alpha = 0,1$, $\gamma = 0,3$, 500 episódios de aprendizagem; *SARSA(λ)*: $\alpha = 0,5$,$\gamma = 0,5$,$\lambda = 0,9$, 700 episódios de aprendizagem; *Actor-Critic*: $\alpha = 0,5$,$\gamma = 1$, $lambda = 0,3$, 100 episódios de aprendizagem.

A Tabela 2 apresenta a média destes valores na SBA *Mundo 4x3*, também considerando 5 execuções por algoritmo, sendo que os parâmetros adotados em cada algoritmo foram: *Q-Learning*: $\alpha = 1$, $\gamma = 1$; *SARSA(λ)*: $\alpha = 1$, $\gamma = 1$,$\lambda = 0,9$; *Actor-Critic*: $\alpha = 1$,$\gamma = 1$, $\lambda = 1$; 500 episódios de aprendizagem em todos os algoritmos.

Nas duas SBAs, os parâmetros de aprendizagem e quantidade de repetições foram selecionados experimentalmente de modo que o agente aprendesse a política ótima. Cabe destacar que a avaliação realizada não tem o intuito de comparar o desempenho dos algoritmos ou encontrar seus parâmetros ótimos, mas sim verificar se a extensão está funcionando adequadamente.

No caso dos algoritmos *Q-Learning* e *SARSA(λ)*, a política aprendida em cada estado corresponde à ação que tem valor máximo de $Q(s, a)$, indicados em negrito nas tabelas. Pode-se verificar em ambas SBAs que o *Q-Learning* faz com que o agente aprenda a política do menor caminho até a célula de destino. Já o *SARSA(λ)* faz com que o agente aprenda uma política diferente do *Q-Learning* na SBA *Cliff Walking*, pois considera a segurança do agente, adotando um caminho que não passa próximo dos estados com recompensa mínima e que representam o penhasco (para evitar risco de queda no penhasco). Essas políticas aprendidas pelo *Q-Learning* e pelo *SARSA(λ)* estão consistentes com o que apontam [Sutton e Barto 2018] para a *Cliff Walking* e [Russel e Norvig 2004] para a *Mundo 4x3*. No *Actor-Critic*, a política aprendida pelo agente em cada estado corresponde à ação que o leva ao estado vizinho de maior utilidade. A partir das tabelas verifica-se que com o *Actor-Critic* o agente também aprende uma política que privilegia a segurança

Estado	*Q-Learning*				*SARSA(λ)*				*Actor-Critic*
(x,y)	Cima	Baixo	Esquerda	Direita	Cima	Baixo	Esquerda	Direita	Utilidade
(0,0)	**-1,42826**	-1,42837	-1,42833	-99,88934	**-1,99636**	-12,05645	-5,23845	-100,00000	-34,44597
(0,1)	-1,42772	-1,42802	-1,42790	**-1,42753**	**-1,99270**	-2,51677	-2,24678	-15,01358	-8,00000
(0,2)	-1,42694	-1,42694	-1,42690	**-1,42684**	-2,30813	-2,32880	-2,01280	**-1,98531**	-7,00000
(1,1)	-1,42611	-98,51834	-1,42668	**-1,42510**	**-1,76315**	-100,0000	-7,02749	-2,57476	-36,99150
(1,2)	-1,42530	-1,42520	-1,42526	**-1,42512**	-3,20844	-8,05625	-2,33142	**-1,98297**	-6,00000
(2,1)	-1,42172	-96,09352	-1,42283	**-1,41700**	**-1,98662**	-99,99982	-12,74462	-19,45179	-47,34018
(2,2)	-1,42094	-1,42080	-1,42122	**-1,42067**	-2,77747	-3,91706	-2,12292	**-1,94073**	-5,00000
(3,1)	-1,40693	-91,22813	-1,40938	**-1,39000**	**-1,97260**	-98,12012	-9,42074	-22,32461	-37,33405
(3,2)	-1,40961	**-1,40914**	-1,40976	**-1,40914**	-2,18766	-11,85778	-2,09680	**-1,88126**	-4,00000
(4,1)	-1,36837	-88,26094	-1,37723	**-1,30000**	-1,90400	-99,32617	-10,46501	**-1,50151**	-34,25671
(4,2)	-1,38097	-1,37830	-1,38035	**-1,37827**	-1,93889	-9,60968	-1,96275	**-1,76251**	-3,00000
(5,1)	-1,26880	**-1,00000**	-1,28172	-1,19679	-1,78433	**-1,00000**	-7,38230	-1,50117	-1,00000
(5,2)	-1,30143	**-1,29590**	-1,30105	-1,30987	-1,96631	**-1,50002**	-1,90512	-1,80811	-2,00000

Tabela 1. Resultados na SBA *Cliff Walking*

Estado	*Q-Learning*				*SARSA(λ)*				*Actor-Critic*
(x,y)	Cima	Baixo	Esquerda	Direita	Cima	Baixo	Esquerda	Direita	Utilidade
(0,0)	**0.840**	0.800	0.800	**0.840**	**0.790**	0.631	0.556	0.359	0,840
(0,1)	**0.880**	0.800	0.840	0.840	**0.841**	0.526	0.683	0.644	0.488
(0,2)	0.880	0.840	0.880	**0.920**	0.541	0.589	0.670	**0.910**	0.952
(1,0)	0.840	0.840	0.800	**0.880**	-0.078	-0.434	**0.691**	-0.001	0,488
(1,2)	0.920	0.920	0.880	**0.960**	0.794	0.777	0.662	**0.952**	0.976
(2,0)	**0.920**	0.880	0.840	0.840	-0.812	-0.252	**0.652**	-0.486	-0,256
(2,1)	**0.960**	0.880	0.920	-1.000	**0.925**	0.297	0.184	-0.800	0.208
(2,2)	0.960	0.920	0.920	**1.000**	0.960	0.865	0.821	**1.000**	1,000
(3,0)	-1.000	0.840	**0.880**	0.840	-1.000	-0.424	**0.298**	-0.305	-0,216

Tabela 2. Resultados na SBA *Mundo 4x3*

de não passar próximo dos estados com recompensa mínima. De forma geral, verifica-se que todas as políticas aprendidas são consistentes com o que cada algoritmo se propõe, evidenciando a consistência da refatoração da extensão.

De forma geral, os resultados obtidos evidenciam que a refatoração da extensão e integração com a biblioteca BURLAP foi bem sucedida. Deste modo, a extensão NetLogo agora disponibiliza três algoritmos de RL para uso em SBAs.

5. Conclusão

Este artigo apresentou a refatoração da extensão NetLogo para aprendizagem por reforço. A extensão foi alterada para utilizar a biblioteca BURLAP já amplamente testada. Também foram incoporados outros dois algoritmos, fazendo com que a extensão suporte *Q-Learning*, *SARSA(λ)* e *Actor-Critic*. Para validar a refatoração, foram implementados duas SBAs: *Cliff Walking* e *Mundo 4x3*. Verificou-se que nessas configurações, os algoritmos *Q-Learning*, *SARSA(λ)* e *Actor-Critic* convergiram às políticas esperadas para esses algoritmos. Esses resultados demonstraram que a refatoração realizada para utilizar a BURLAP não introduziu inconsistências na extensão e permitiu a incorporação bem-sucedida dos algoritmos *SARSA(λ)* e *Actor-Critic*.

Como sugestão para trabalhos futuros, recomenda-se realizar uma avaliação da extensão em diferentes SBAs com o objetivo de demonstrar sua aplicabilidade em diversos cenários de aprendizagem por reforço. Além disso, seria interessante incorporar outros algoritmos de aprendizagem disponíveis na BURLAP, a fim de expandir as opções

e possibilidades da extensão. Essas melhorias promoveriam a versatilidade e o potencial de aplicação da extensão em um contexto mais amplo.

Os autores agradecem a FAPESC pelo apoio financeiro recebido (TO2021TR882 e TO2023TR246).

Referências

Bazzanella, E. e Santos, F. (2021). Does a q-learning netlogo extension simplify the development of agent-based simulations? In *Anais do XV Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCações*, pages 1–12, Rio de Janeiro.

Bonabeau, E. (2002). Agent-based modeling: methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(3):7280–7287.

CoMSES (2020). CoMSES Catalog. `https://catalog.comses.net/visualization/`, Acesso em: Jul/2020.

Epstein, J. e Axtell, R. (1996). Growing artificial societies: social science from the bottom up. *Brookings Institution Press*.

Konda, V. e Tsitsiklis, J. (2000). Actor-critic algorithms. *Advances in neural information processing systems*, pages 1008–1014.

Kons, K. (2019). *Biblioteca Q-Learning para desenvolvimento de simulações com agentes na plataforma NetLogo*. Trabalho de conclusão de curso, Universidade do Estado de Santa Catarina (UDESC).

Macal, C. M. e North, M. J. (2010). *Introduction to agent-based modeling and simulation*. Springer Science & Business Media.

MacGlashan, J., Loftin, R., Littman, M. L., e Roberts, D. L. (2018). BURLAP: Brown-UMBC Reinforcement Learning and Planning. `burlap.cs.brown.edu/`.

Macy, M. W. e Willer, R. (2002). The factors affecting cooperation in a social dilemma: A multiagent simulation. *Computational & Mathematical Organization Theory*, 8(3):187–207.

Peters, J. e Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

Rummery, G. e Niranjan, M. (1994). On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*.

Russel, S. e Norvig, P. (2004). *Inteligência Artificial*. Rio de Janeiro: Campus, 2 edition.

Sklar, E. (2007). Software review: Netlogo, a multiagent simulation environment. *Journal of Artificial Life*.

Sutton, R. S. e Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tisue, S. e Wilensky, U. (2004). Netlogo: Design and implementation of a multi-agent modeling environment. In *Proceedings of agent*, volume 2004, pages 7–9. Springer.

Wang, X.-S., Cheng, Y.-H., e Yi, J.-Q. (2007). A fuzzy actor–critic reinforcement learning network. *Information Sciences*, 177(18):3764–3781.

Watkins, C. J. C. H. e Dayan, P. (1992). Q-learning. *Machine learning*, 33(3–4):279–292.

Wilensky, U. (1999). NetLogo. `http://ccl.northwestern.edu/netlogo/`.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.